Model based dynamics compensation for humanoid robots

Technische Universiteit
Eindhoven
University of Technology

Department of Mechanical Engineering
D&C

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Author
T.T.B.Dorussen,
Mechanical Engineering

Date
30 August 2013

Model Based Dynamics Compensation for humanoid Robots DC 2013.049

Contents

List of Symbols

Symbol	Definition	Unity
CoM	Centre of Mass	[m]
D	Variable	[m^2]
H	Centripetal, Coriolis and Gravity terms	[N]
J_c	Jacobian of Lagrange Multipliers	[-]
K/P	Posive definte weighting matrix	[-]
l	Length of link	[m]
L	Lagrangian of convex minimization problem	[-]
LB	Left bound	[m]
M	Mass matrix	[kg]
q	Angle/position of coordinates	[m or rad]
$\dot{q}$	Velocity of coordinates	[m or rad/s]
$\ddot{q}$	Acceleration of coordinates	[m or rad/s^2]
Q	Part of QR-decomposition of J_c	[-]
R	Part of QR-decompostion of J_c	[-]
RB	Right bound	[m]
S	Generalized Force Direction	[m]
t	time	[t]
w	Jacobian of W	
W	Transposed Jacobian of Lagrange Multipliers	[-]
ZMP	Zero Moment Point	[m]
α	Parameter	[-]
θ	Angle	[rad]
λ	Lagrange Multiplier/Contact forces	[N]
τ	Torques	[N/m]

Summary

This study concerns the control of the walking motion of a biped humanoid robot during its double support phase, where both feet are in contact with the ground. In this situation the robot is over-actuated, which means there are many sets of actuator torques and contact forces that satisfy the general equation of motion. The goal of this study is to design a controller that simultaneously produces a desired trajectory and a desired contact force profile; a smooth transition period between single support on the left leg to single support on the right leg. This is obtained by gradually shifting the contact forces from totally on the left foot to totally on the right foot, so that at the end of the double support phase, the left leg can be lifted and the walking motion can be continued.

A new method is developed based on decomposition of the general equation of motion with additional artificial constraints to obtain the shifting contact forces. With the newly designed method a controller is built for a 2D robot with a 3^{th} order trajectory. The produced contact forces indeed gradually shift from the left to the right foot. With the calculated contact forces the actuator torques are also computed. A dynamic simulation is performed to check if the set of contact forces and torques produce a trajectory that matches the desired trajectory. The trajectory does follow the desired trajectory given a low simulation tolerance (order 10^{-12}). The difference between desired and real trajectory was of order 10^{-3}; it can be assumed that the torques and contact forces given by the controller produce a trajectory which matches the desired trajectory.

For the 2D example this method works without any disturbances. But before this method can be implemented on a real 3D robot some issues have to be resolved. To supress the disturbance a feed-back controller has to be added to the model; a simple joint PD-controller is probably already sufficient. In 3D the inverse kinematics and the ZMP become far more complex and the methods used for 2D in this study are possible no longer sufficient.

1 Introduction

Movies like ‘I robot’ and ‘The terminator’ are science-fiction based movies, but are becoming more and more realistic, because of the extensive research in robotics in the last decades. Human-like robots, in the sense they can move like humans, are called humanoid robots. In the (near) future these humanoid robots can be used to substitute people in various quite simple household tasks like lawn mowing and vacuuming but also tasks in industry, services and care. They also can be useful in physically demanding and dangerous work.

The Dynamics and Control group of the Eindhoven University of Technology has a activity group, Tech United, which focuses on humanoid robots [1]. The humanoid robot it currently has, is called TUlip. This robot uses an algorithm that enables it to walk slowly but is relatively robust against disturbances [2]. This study focuses on a problem that still has to be addressed; the control of the robot when the robot it is supported by both feet (double support). The single supported TUlip can be assumed as a series of rigid bodies, for which classical model based dynamics compensation algorithms can be used [3]. However, in the double support phase TUlip loses degrees of freedom and becomes an over-actuated system [2]. A smart choice between the actuator torques should be made, which should provide a smooth transition from single support on the left leg to single support on the right leg. It resembles the motion of first using solely the left foot, to using both feet and ending solely balancing on the right foot after which single support control can be used to continue the walking motion. This motion can be provided by controlling the contact forces. If the single support algorithm is used for the whole system, a jump in the torques occurs at the moment the second foot touches the ground. This is because the algorithm switches from based totally on the left foot, to totally based on the right foot. This jump causes a shock, which is not desirable.

The goal of this study is to provide a controller for the double support phase that controls the gradual shift in the contact forces while providing torques that makes the robot follow a desired trajectory. The objective of this study is briefly stated in the following statements [2]:

- review existing (humanoid) dynamics compensation literature (Ch. 3.1)
- design a model based feed forward gravity compensation algorithm that can smoothly handle single as well as double support situations (Ch. 3.2)
- verify the algorithm in simulation (Ch. 4)

In Chapter 2 of the report the robot TUlip will be described more extensively. In Chapter 3 an existing algorithm for control of the double support phase is explained and a newly designed method is developed. Chapter 4 introduces a more simplified 2D-model of a humanoid robot which will be used in the rest of the report. For this 2D-example a controller is designed according to one of the proposed algorithms in Chapter 3. These results are presented in Section 4.4 and 4.5. In Section 4.6 various topics surrounding this method are discussed, such as the difference between 2D and 3D and adding feedback to the controller. In the appendix the full Matlab scripts used to compute the 2D model and its controller are included.

2 Modelling a humanoid robot

2.1 TUlip

The Eindhoven University of Technology has collaborated with the other Universities of Technology in the Netherlands (Delft and Twente) and Philips to develop a humanoid robot. A humanoid robot is a robot that resembles the human body and mimics its kinematics. That means it has two legs consisting of a ankle, a knee and a hip joint. This particular humanoid robot is called TUlip, a biped walking robot, and is used as an experimental platform for research on walking, dynamical analysis, control and artificial intelligence of humanoid robots [4].

Figure 1: Photo of TUlip [1].

TUlip has 16 degrees of freedom (DOF), all revolute, of which six in each leg (three per hip). It also has arms and 2 cameras built into the head, which are not relevant for this study. TUlip's limbs consist out of black anodized aluminium components protected by a soft foam and tough aramid composite material with actuators in each joint. TUlip works on GHz computer, which is fed by a battery pack or is connected to an auxiliary power supply, that also feeds the actuators. The robot can last for about half an hour on batteries before it runs out of power. The robot weights around 24 kilos excluding batteries [5].

TUlip also participated in RoboCup 2013, a football tournament for robots, in Eindhoven; it lost in the semi-finals of the humanoid adult size class with 0-3 from HuroEvolution AD of the National Taiwan University of Science and Technology [6]. With the current controller it was able to walk, but it was too slow to get to the ball within the 2.5 minute time limit. Besides the fun, this tournament gives universities the chance to compare humanoid robots and their techniques, so they can learn from each other, stimulate the progress in the field even more and lead to various breakthroughs.

2.2 Theoretical model

To describe the kinematics of a dynamical system various methods are proposed. One of the most generic methods is the Denavit-Hartenberg convention [7]. This method describes the position of all coordinate frames of the system with 4 parameters with respect to the previous coordinate frame: link length, link twist, link offset and joint angle. TUlip is modelled as a chain of thirteen rigid bodies with point mass and actuators in each joint. The two feet have each 4 contact points. The model of TUlip is shown according to the Denavit-Hartenberg convention in Figure 2.

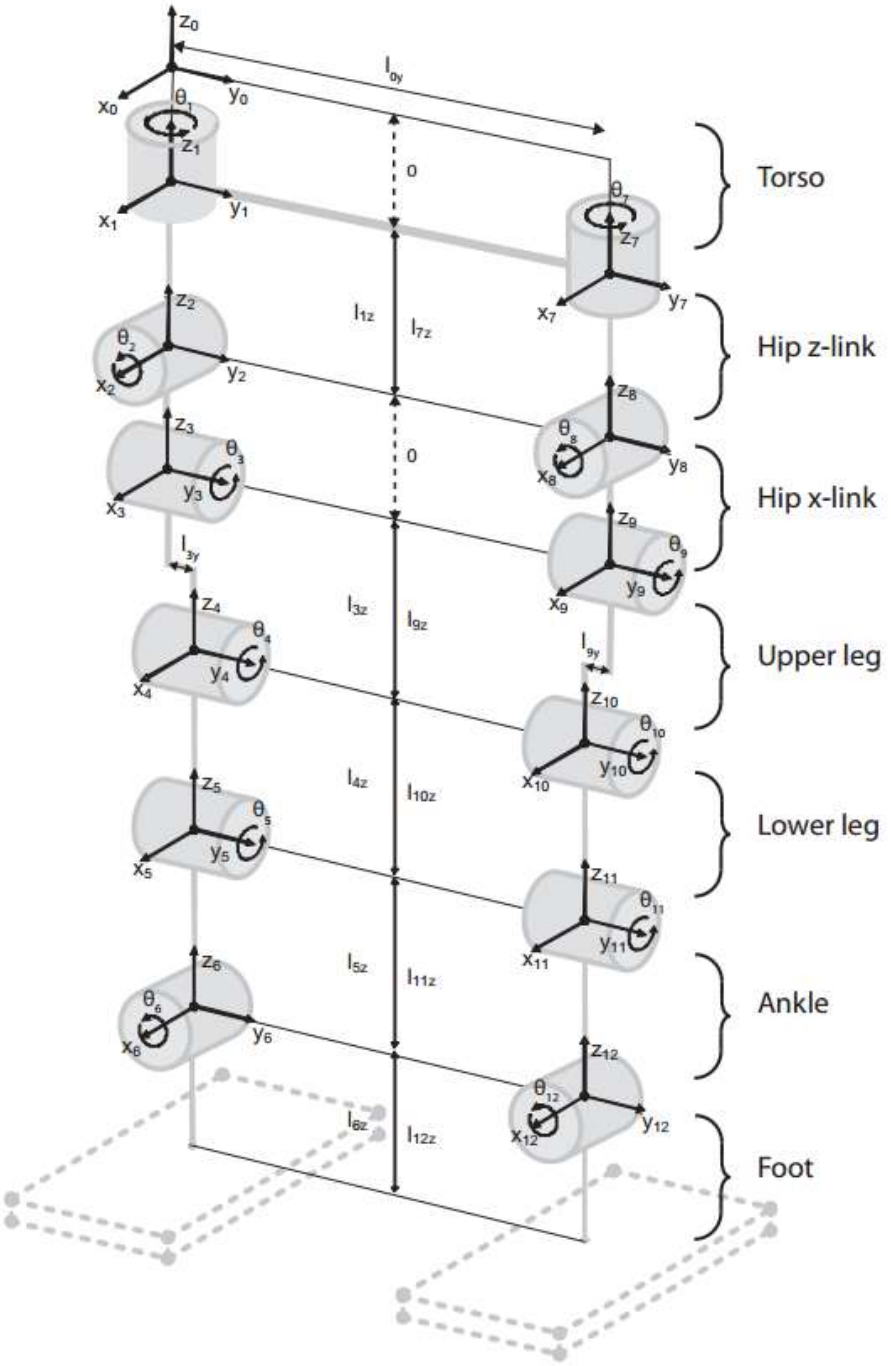

Figure 2: Model of TUlip according to Denavit-Hartenberg convention [5].

Consistently using this convention produces the equation of motion of the total system, which in its most generic form is described by [3]:

$$M\ddot{q} + H(\dot{q}, q) = S\tau + J_c^T \lambda, \tag{2.1}$$

where M is the inertia matrix, H is the combined vector of centripetal, Coriolis and gravity terms, while the torques applied on the joints are described by τ. The matrix S represents the generalized force directions of the torques. Constraints are locations of the system that are in environmental contact, so external forces or torques are applied; there is no velocity and no acceleration at these locations. If the system is constrained in k points, k Lagrange multipliers λ have to be added to the system. In the case of a robot with both feet on the ground, λ represents the contact forces with the ground, while J_c is the Jacobian of these constraints. If the position of these constraint locations are described by x_c, the next equations should also hold [8]:

$$\dot{x_c} = J_c \dot{q} = 0 \tag{2.2}$$

$$\ddot{x_c} = J_c \ddot{q} + \dot{J_c} \dot{q} = 0 \tag{2.3}$$

For this study on a humanoid robot, the matrices M, H, S and J_c are already computed on previous studies [4] and are adopted as such, so no further analysis regarding these matrices is done.

In this study the humanoid robot has to follow a desired trajectory, while gradually changing from totally leaning on the left foot to fully leaning on the right foot. In symbolic expressions this means that suitable actuator torques τ have to be found to follow a certain desired trajectory q, while also controlling the contact forces λ. The contact forces have to be controlled such that the vertical contact forces are always positive, because else this contact point would make a jump. The horizontal contact forces represent the friction force and are not of importance; these contact forces do not have to be controlled. Additionally, the vertical contact forces also have to change gradually from the left foot to the right foot during the double support phase. That is because if all the contact force is supported on the right foot at the end of the double support phase, the left foot can be lifted from the floor and then the single support controller can be applied and the robot can continue with its walking cycle. So the controller has to control both the desired trajectory and the vertical contact forces.

3 Methods to design feed forward algorithm

There are several methods to design a suitable feed-forward controller for the double support phase. First, a method is presented, which is available in literature. The second method that will be presented is a newly designed method and this will be used in the remainder of the report, because it exactly fits the goal of this study where it gradually shift the contact forces from left to right, while the other method does not fulfil that purpose.

3.1 Inverse dynamics via orthogonal decomposition

The equation of motion as given in (2.1) is quite complex because in general both the actuator torques and the contact forces are unknown. Therefore, a qr-decomposition of the Jacobian of the constraints J_c is proposed in the paper of Mistry, Buchli and Schaal [8]:

$$J_c^T = [Q_c \quad Q_u] \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{3.1}$$

This divides the equation of motion into two separate parts, of which the second equation is independent of the contact forces.

$$Q_c^T(M\ddot{q} + H) = Q_c^T S + R\lambda \tag{3.2}$$

$$Q_u^T(M\ddot{q} + H) = Q_u^T S \tag{3.3}$$

With these formulas, first the control torques can be computed; after that the contact forces are calculated.

$$\tau = (Q_u^T S)^+ Q_u^T(M\ddot{q} + H) \tag{3.4}$$

$$\lambda = R^{-1} Q_c^T(M\ddot{q} + H - S\tau) \tag{3.5}$$

The plus sign in (3.4) indicates a pseudo inverse, which can be used for non-square matrices. To minimize the norm of the torque vector, the Moore-Penrose pseudo inverse [8] can be used:

$$A^+ = P^{-1}A^T(AP^{-1}A^T)^T \tag{3.6}$$

in which P is a positive definite weighting matrix that prioritizes specific torque loads. This is useful when the system is over-actuated because it chooses one of these solutions based on the weighting matrix.

For these equations to hold, the motion has to be in the null space of the constraints; in other words, the QR decomposition only hold if (2.2) and (2.3) hold.

This method first computes the forces in the joint and after that the contact forces. The goal in this study is to control the contact forces and a quick study revealed that only with the weighting matrix P as parameter, no real influence on the contact forces can be performed. Therefore, this method does not fit for control of a humanoid robot during the phase where both feet of the robot are on the ground.

3.2 Inverse dynamics via decomposition and artificial constraints

In this study the distribution of the contact forces is of importance as well as following the trajectory; based on this fact a new method is developed in the following Section.

3.2.1 Decomposition

The general equation of motion that was stated in (2.1) consists of multiple equations. Taking a closer look at these equations, reveals that k equations are independent of the actuator torque and are only dependent on the contact forces. With this fact the general equation can be separated into two equations of which one is only dependent on the contact forces (comparable to the qr-decomposition); matrix S_1 and S_2 are designed to obtain these two separate equation. Both matrices have the same number of rows and columns. The number of rows is given by the total number degrees of freedom of the system (DOF) while the number of columns is given by the amount of torque actuators (n).

$$S_1 = \begin{bmatrix} I_{kxk} \\ O_{(DOF-k)x(n-k)} \end{bmatrix}, \qquad S_2 = S$$

With these new matrices also the mass matrix M, as well as H and J_c can be separated in two parts.

$$M_1 = S_1^T M, \qquad H_1 = S_1^T H, \qquad J_{c1} = J_c S_1$$

$$M_2 = S_2^T M, \qquad H_2 = S_2^T H, \qquad J_{c2} = J_c S_2$$

The general equation of motion can now be written as the separate expressions in which the desired trajectory is used for the position, velocity and acceleration of the coordinates:

$$M_1 \ddot{q}_d + H_1(\dot{q}_d, q_d) = J_{c1}^T \lambda \tag{3.7}$$

$$M_2 \ddot{q}_d + H_2(\dot{q}_d, q_d) = \tau + J_{c2}^T \lambda \tag{3.8}$$

From the first equation the contact forces can be calculated and with these contact forces the second equation computes the torques that follow the desired trajectory. Now only a controller has to be built for the first equation that controls the contact forces.

3.2.2 Parameter α

Equation (3.7) computes the contact forces. This equation only has unknown contact forces. The goal was to control the system such that also the vertical contact forces gradually shift from the left foot to the right foot. The parameter α is introduced to obtain this shift; for $\alpha = 0$, the vertical forces on the right foot should be 0, while for $\alpha = 1$ the vertical forces on the left foot should be 0. Contact forces λ for an example with 3 contact points per foot are defined as follows:

- λ_1: Horizontal contact force of left foot
- λ_2: Vertical contact force at the toe of the left foot
- λ_3: Vertical contact force at the heel of the left foot
- λ_4: Horizontal contact force of right foot
- λ_5: Vertical contact force at the toe of the right foot
- λ_6: Vertical contact force at the heel of the right foot

To obtain this shift a relation between the contact forces and α should be realized. The parameter α resembles the factor between the vertical contact forces on the right foot compared to the left foot:

$$\frac{\lambda_5+\lambda_6}{\lambda_2+\lambda_3+\lambda_5+\lambda_6} = \alpha \tag{3.9}$$

As stated before in Section 2.3, the contact forces in vertical direction per definition should be positive for the feet to stay on the ground. Therefore, (3.9) is not sufficient, because it is for example still possible to have 1 negative contact force, while the others are positive. So the requirement that all the vertical contact forces have to be positive, is not satisfied by this equation. If (3.9) is split into two separate formulas, the same factor α holds for the combined contact forces.

$$\frac{\lambda_5}{\lambda_2+\lambda_5} = \alpha \tag{3.10}$$

$$\frac{\lambda_6}{\lambda_3+\lambda_6} = \alpha \tag{3.11}$$

With these equations it is not possible to find 1 negative contact force. It still is possible to find for example negative λ_2 and λ_5 that satisfies these equations. To counteract this, the Moore-Penrose pseudo inverse is used, which will be explained more extensively in Section 3.2.4. With these two formula applied on an example, the contact forces are plotted as function of α in Figure 3.

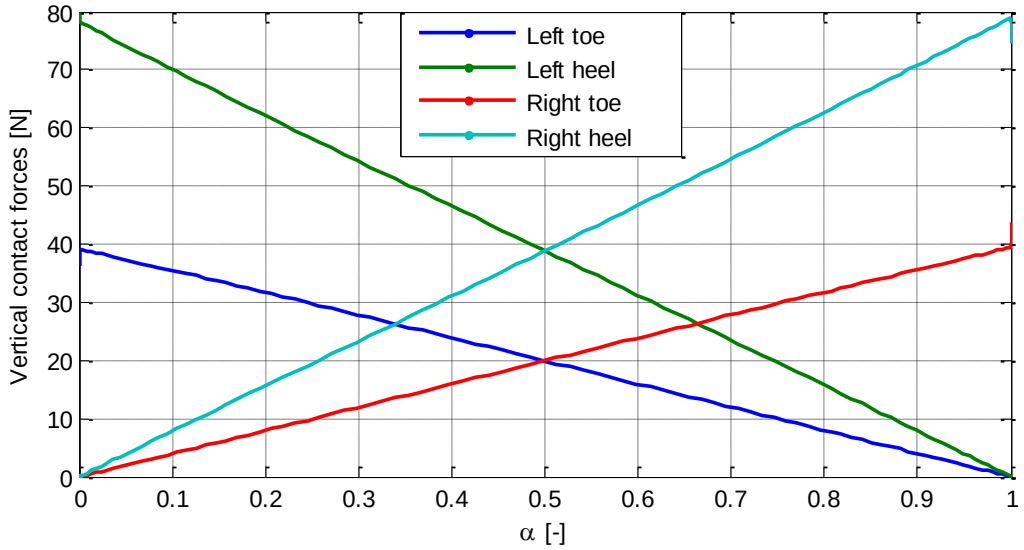

Figure 3: Vertical contact forces as function of α.

This figure shows all the contact forces are positive and shift linearly as function of α from the left foot to the right foot. Now these relation have to be rewritten into a different form so they can be added to (3.7):

$$-\alpha\lambda_2 + (1-\alpha)\lambda_5 = 0 \tag{3.12}$$

$$-\alpha\lambda_3 + (1-\alpha)\lambda_6 = 0 \tag{3.13}$$

These two constraints now can be added to (3.7):

$$c_1 = [0 \quad -\alpha \quad 0 \quad 0 \quad 1-\alpha \quad 0]$$

$$c_2 = [0 \quad 0 \quad -\alpha \quad 0 \quad 0 \quad 1-\alpha]$$

$$A = \begin{bmatrix} J_{c1}^T \\ c_1 \\ c_2 \end{bmatrix}, \qquad b = \begin{bmatrix} M_1\ddot{q} + H_1 \\ 0 \\ 0 \end{bmatrix}$$

The contact forces can now be obtained by solving (3.14), which still is over-actuated, so infinite number of solutions are possible:

$$b = A\lambda \tag{3.14}$$

For the relations to hold α should always be between 0 and 1. Therefore, the parameter α is saturated between 0 and 1.

3.2.3 Zero Moment Point

Now the parameter α is introduced, a physical meaning should be attached to the parameter. It already resembles the factor of vertical contact forces between right and left foot as shown in Figure 3. If the system is in rest, the vertical contact forces are solely the result of gravity acting on the system. That means α cannot be chosen at will, but follows out of the gravity forces which are affected by the positions of the joints. In mechanics, the calculations of these gravity forces are simplified by using the centre of mass (CoM), which is a unique point that represents the accumulated gravity force in a point mass. For the system to be feasible, the point that resembles the factor between the contact forces has to match witch the location of the CoM. The CoM is described by:

$$CoM = \frac{\sum m_i x_i}{\sum m_i} \tag{3.15}$$

If the system is in motion, which is the case for the robot, the CoM is no longer suitable, because it does not take velocity and accelerations into account. Its dynamical equivalent should be used, which is called the Zero Moment Point (ZMP) [9]. In the computation of the ZMP the velocities and accelerations are also taken into account:

$$ZMP = \frac{\sum m_i\left(x_i(\ddot{y}_i+g_y) - y_i(\ddot{x}_i+g_x)\right)}{\sum m_i(\ddot{y}_i+g_y)} \tag{3.16}$$

If there is no motion, the ZMP is equal to the CoM. For every moment in the trajectory of the robot (position, velocity and acceleration are known) the ZMP can be calculated. The robot shifts its ZMP from the middle of its left foot (LB) to the middle of its right foot (RB); this is a design choice. Now the parameter α can be calculated with:

$$\alpha = \frac{\text{ZMP}-\text{LB}}{RB-LB} \tag{3.17}$$

3.2.4 Computing solutions

With the formulas given in the previous sections, the contact forces and torques can be calculated from the equation of motion with addition of the constraints of Section 3.2.2:

$$\lambda = A^{+}b \tag{3.18}$$

$$\tau = M_2\ddot{q} + H_2(\dot{q}, q) - J_{c2}^{T}\lambda \tag{3.19}$$

If A is not full rank, the inverse cannot be computed. Instead of the inverse, the Moore-Penrose pseudo inverse ($A^{+} = A^{T}(AA^{T})^{-1}$) can be used [8]. This inverse finds the minimal norm of the contact force vector that satisfies the equation. This means that if there is a solution with positive contact forces, the Moore-Penrose pseudo inverse will find these values, because the positive values are the minimal norm of the vector. That is because the 4 vertical contact forces must accumulate to a certain positive value. One negative contact force should be compensated by bigger positive value, which leads to a bigger norm vector.

If more emphasis is desired on a certain contact force, then the weighted Moore-Penrose pseudo inverse can be used:

$$A^{+} = P^{-1}A^{T}(AP^{-1}A^{T})^{-1} \tag{3.20}$$

In this study, this is not the case, therefore the weighing matrix P is the unity matrix, which produces the pseudo inverse.

4 A 2D-example

The complete model of TUlip in 3D has many coordinates and requires a long computational time. This complicates the methods stated in Chapter 3. Therefore a simpler 2D-model is used with less DOFs and less computational time. It resemble the 3D model in the basic theoretical aspects (equation of motion, kinematics, torques, contact forces).

4.1 Properties

The 2 dimensional representation of the robot has 6 joints: 3 in each leg (hip, knee and ankle). Three additional coordinates (x, y and q_1) are required to describe the position and orientation of the robot in space. This model is used to calculate the feed-forward controller and therefore its properties are explained in detail. As stated before the robot has 9 coordinates in total:

- x : Horizontal position of torso
- y : Vertical position of torso
- q_1 : Rotation of torso
- q_2 : Rotation of left hip joint
- q_3 : Rotation of left knee joint
- q_4 : Rotation of left ankle joint
- q_5 : Rotation of right hip joint
- q_6 : Rotation of right knee joint
- q_7 : Rotation of right ankle joint

The lengths and weights of the body parts are given in the next table. Left and right leg have the same properties.

Table 1: Weight and measurements of the body part of the robot.

Body part	Length [m]	Weight [kg]
Torso	0.5	5
Upper leg (left/right)	0.5	2
Lower leg (left/right)	0.5	1
Foot (left/right)	0.1	0.5

For each body part the centre of gravity is assumed in the middle of the part; just like in the 3D model, the parts are modelled as a point mass. There are 6 actuators in total; one at each joint (hip, knee and ankle on both left and right leg). Each foot has 3 constraints with the ground; 1 in horizontal direction (friction force) and 2 in vertical direction (normal force), which are modelled at the heels and toes. The model as described here is presented in Figure 4. The inertia matrix, gravity terms etcetera were already calculated by Pieter van Zutven in the Matlab file described in Appendix A.1.

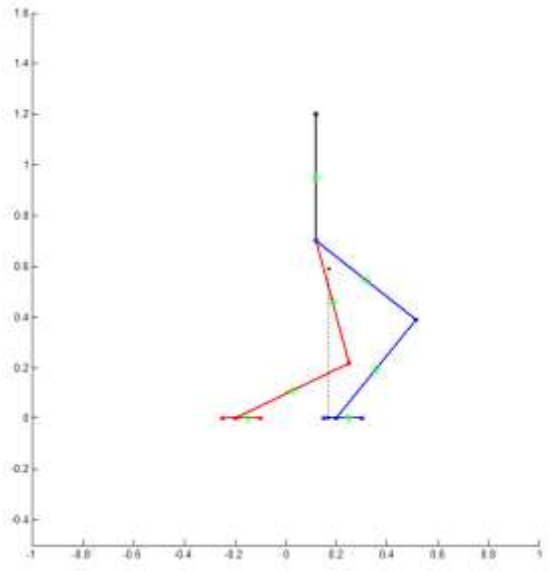

Figure 4: 2D model of humanoid robot.

In this study the trajectory with both feet on the ground is of importance; therefore both feet are modelled fixed on the ground at the ankles at $LB = -0.2\,m$ (left foot) and $RB = 0.2\,m$ (right foot), while both feet have to be horizontal with the ground. These values are defined with respect to a fixed frame in space. This fixed feet will be used in the calculations of the inverse kinematics in Section 4.3.

4.2 3th order trajectory

For the desired movement of the robot a trajectory in one points of the robot has to be designed. Subsequently, with inverse kinematics the motion in all the coordinates can be calculated. Because the ZMP, explained in the previous chapter, on which α is based, has enormous computations if you want the calculate the inverse kinematics, it is more convenient to base the trajectory not on the ZMP. Therefore it is chosen to base the trajectory on the position of the hip. The vertical direction of the hip y is chosen as a constant (0.7 m) and only a trajectory in horizontal direction is computed. This is not a human-like walking movement, but sufficient to evaluate the feed forward controller. A third order trajectory is described by [10]:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{4.1}$$

With a known position and velocity at time 0 and after 2 seconds, the parameters can be solved by:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & t_f & {t_f}^2 & {t_f}^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_f \\ \dot{q}_f \end{bmatrix} \tag{4.2}$$

For the position at t_0 and t_f parameter α has to fluctuate between 0 and 1. With trial and error the positions are determined at $q_0 = -0.33$ and $q_f = 0.12$, with $\dot{q}_0 = \dot{q}_f = 0$. For coordinate x this produces the trajectory as shown in Figure 5.

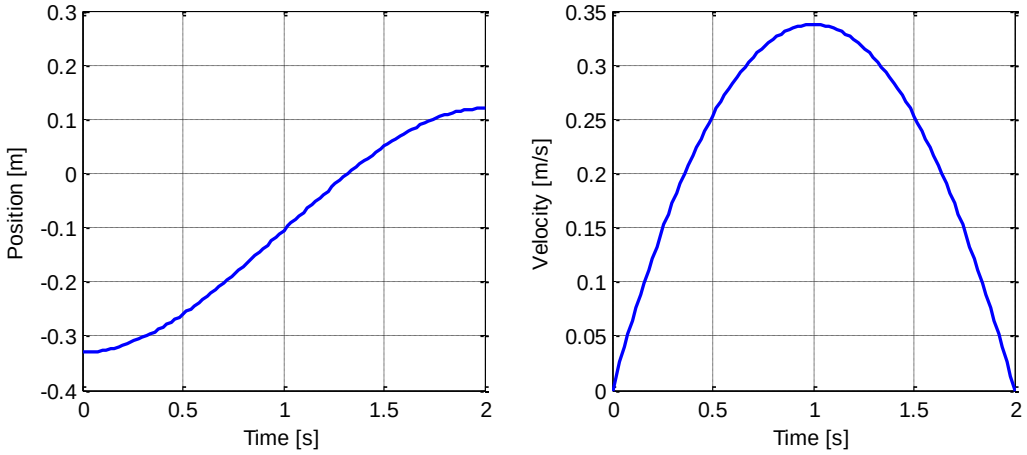

Figure 5: Position and velocity of x, computed by 2nd order trajectory.

4.3 Inverse kinematics

With the trajectory of x, the constant y, and the fixed positions of both feet, the desired angles of coordinates q can be determined by inverse kinematics. First, the torso always has to be vertical, which means $q_1 = 0$ and constant.

To calculate the rotations in the hip and knee (q_2, q_3, q_5 and q_6), both legs are assumed as a two-link planar arm [11]. The inverse kinematics used for the left leg are (the right leg is not explicitly stated because it is identical, except that θ_3, θ_4 should be used and variable $D = (x - 0.2)^2 + y^2$):

$$D = \cos(\theta_2) = (x + 0.2)^2 + y^2 \tag{4.3}$$

$$\sin(\theta_2) = \pm\sqrt{1 - D^2}\ (elbow\ up\ or\ elbow\ down) \tag{4.4}$$

$$q_2 = 1.5\pi - atan\frac{y}{x} - atan\frac{l_2\sin(\theta_2)}{l_3 + l_2\cos(\theta_2)} - atan\frac{\sin(\theta_2)}{\cos(\theta_2)} \tag{4.7}$$

$$q_3 = atan\frac{\sin(\theta_2)}{\cos(\theta_2)} \tag{4.8}$$

$$q_5 = 1.5\pi - atan\frac{y}{x} - atan\frac{l_5\sin(\theta_4)}{l_6 + l_5\cos(\theta_4)} - atan\frac{\sin(\theta_4)}{\cos(\theta_4)} \tag{4.9}$$

$$q_6 = atan\frac{\sin(\theta_4)}{\cos(\theta_4)} \tag{4.10}$$

Only the coordinates of the ankles (q_4 and q_7) are left. Because both feet have to be horizontal, while the torso is vertical, the expressions are quite simple to find:

$$q_4 = 0.5\pi - q_2 - q_3 \tag{4.11}$$

$$q_7 = 0.5\pi - q_5 - q_6 \tag{4.12}$$

For calculating the velocity and acceleration the derivatives of the desired joint angles are taken. The function file where all the velocities and accelerations are computed is presented in Appendix A.2.

The 3th order trajectory combined with the inverse kinematics of the model, produce a series of coordinates, which resemble a movement of the robot from left to right, while its feet stay on the ground. Figure 6 shows 4 snapshots of this motion.

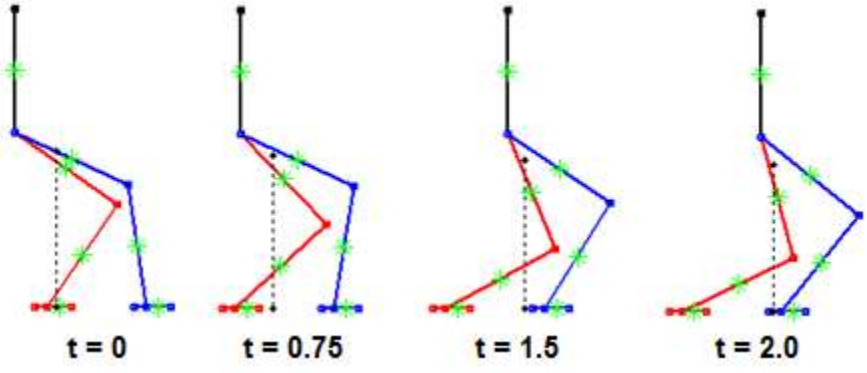

Figure 6: Part of walking cycle 2D robot with both feet grounded.

The desired trajectory given in Section 4.2 is applied to this system and produces the following desired joint orientations, velocities and accelerations calculated according to the relations of inverse kinematics.

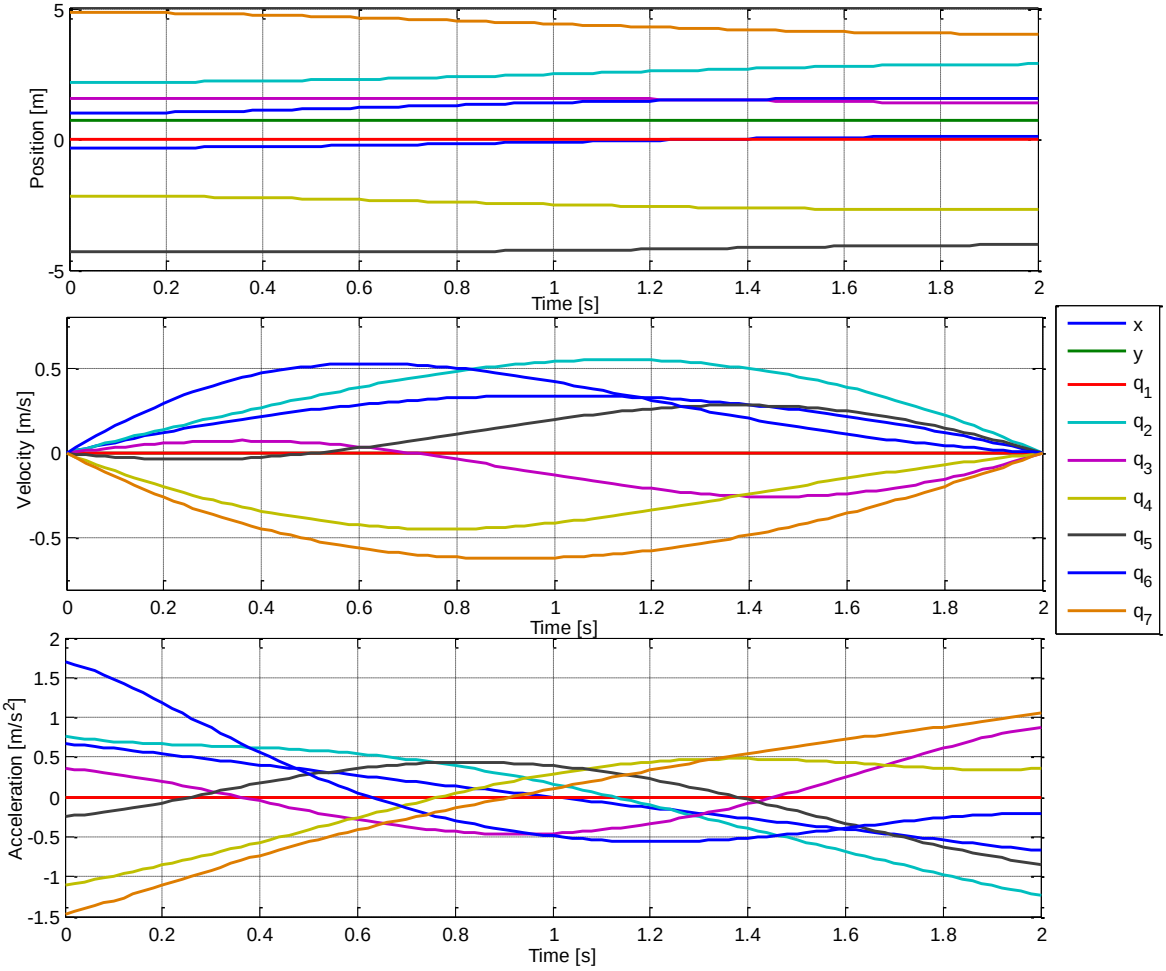

Figure 7: Desired position, velocity and acceleration in every coordinate (see begin of 4.1) during double support phase of walking cycle.

4.4 Contact forces and torque

With the equations mentioned in Section 3.3 applied on the 2D-example described in Chapter previous of chapter 4, the controller for the 2D-system can be calculated with Matlab (Appendix A.3). The robot has 9 coordinates, 6 actuators and 6 contact forces with the ground. With the desired kinematics of the robot computed in Section 4.3 the contact forces can be calculated for every step. As Figure 8 shows, the horizontal contact forces (λ_1 and λ_4) are identical, small and partly negative, but are not of any concern for this study. The other contact forces are all positive and shift gradually from the left foot to the right foot over the full range of α. At $t = 0$ there is no force on the right foot, while at $t = t_f$ no vertical force acts on the left foot. Now this foot can be lifted to continue with the walking motion.

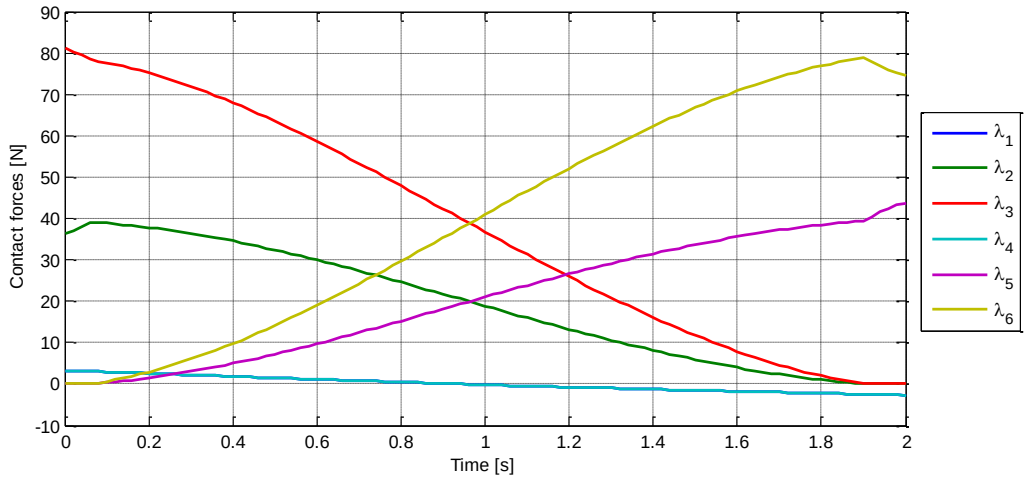

Figure 8: Contact forces acting on the robot as function over time.

With the contact force calculated, only (3.19) remains, which produces the torques as shown in Figure 9, with the torques defined as such:

- τ_1: Torque at left hip
- τ_2: Torque at left knee
- τ_3: Torque at left ankle
- τ_4: Torque at right hip
- τ_5: Torque at right knee
- τ_6: Torque at right ankle

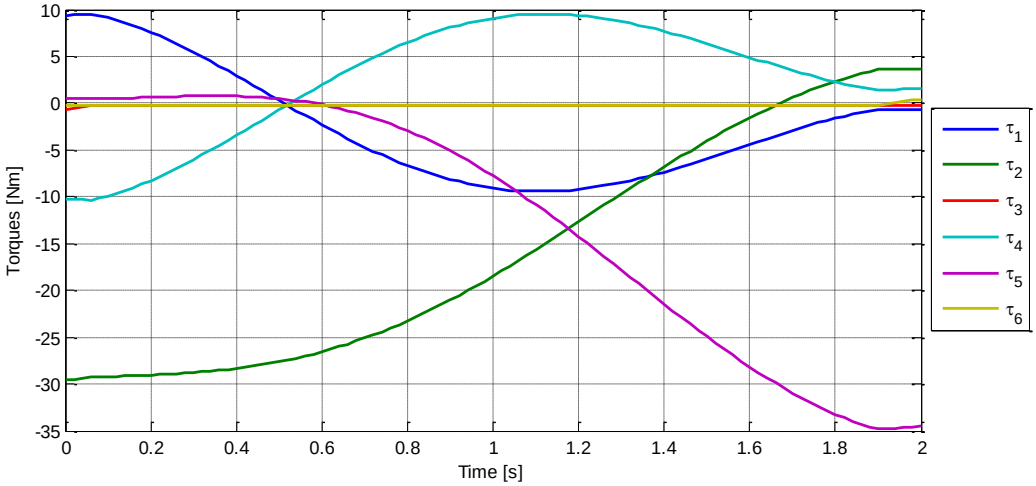

Figure 9: Torques acting on the robot as function over time

4.5 Real kinematics

With the calculated torques a dynamic simulation can be performed, which produces kinematics that of course should follow the desired trajectories. Therefore the following system is implemented in Matlab (Appendix A.4) and numerically integrated [3]:

$$\lambda = (W^T M^{-1} W)^{-1}(W^T M^{-1}(H - S\tau) - w) \tag{5.1}$$

$$\ddot{q} = M^{-1}(S\tau + W\lambda - H) \tag{5.2}$$

Now for every step the system is solved as function of the torques. With a small simulation tolerance (10^{-12}) the evolution of the system coordinates is computed and shown in Figure 10, next to the desired trajectories, while the error function is plotted in Figure 11.

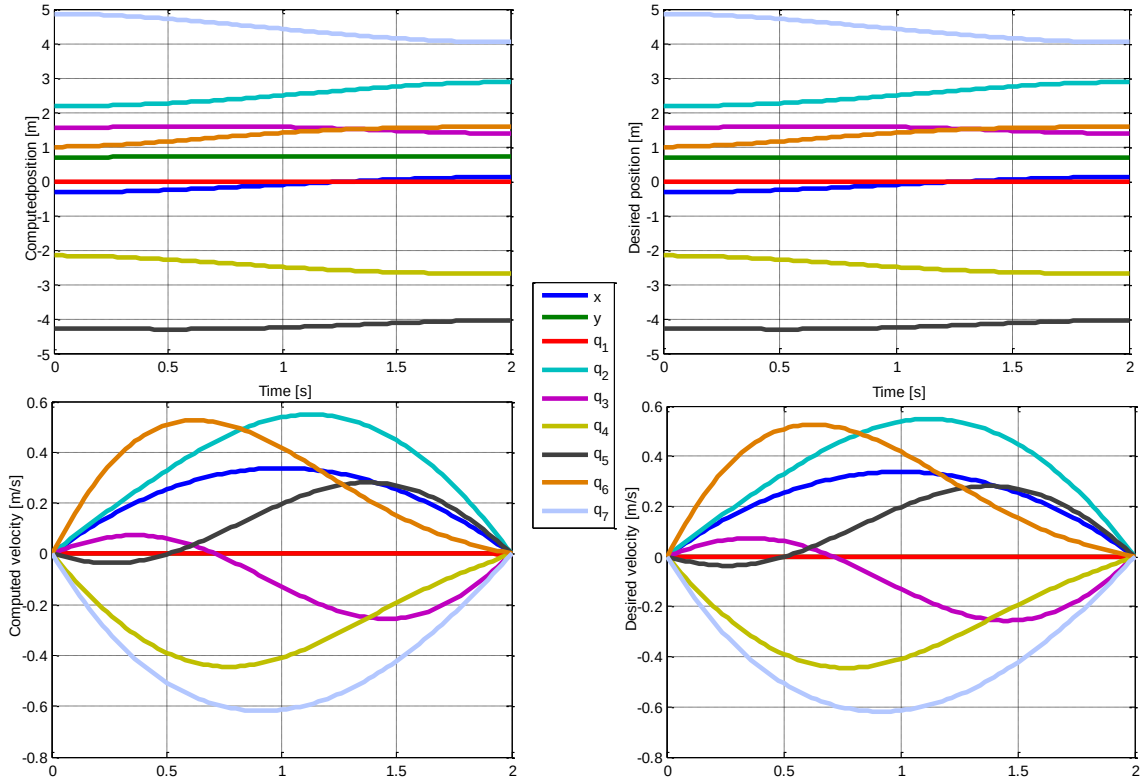

Figure 10: Computed kinematics (left) compared to the desired kinematics (right) as function of time.

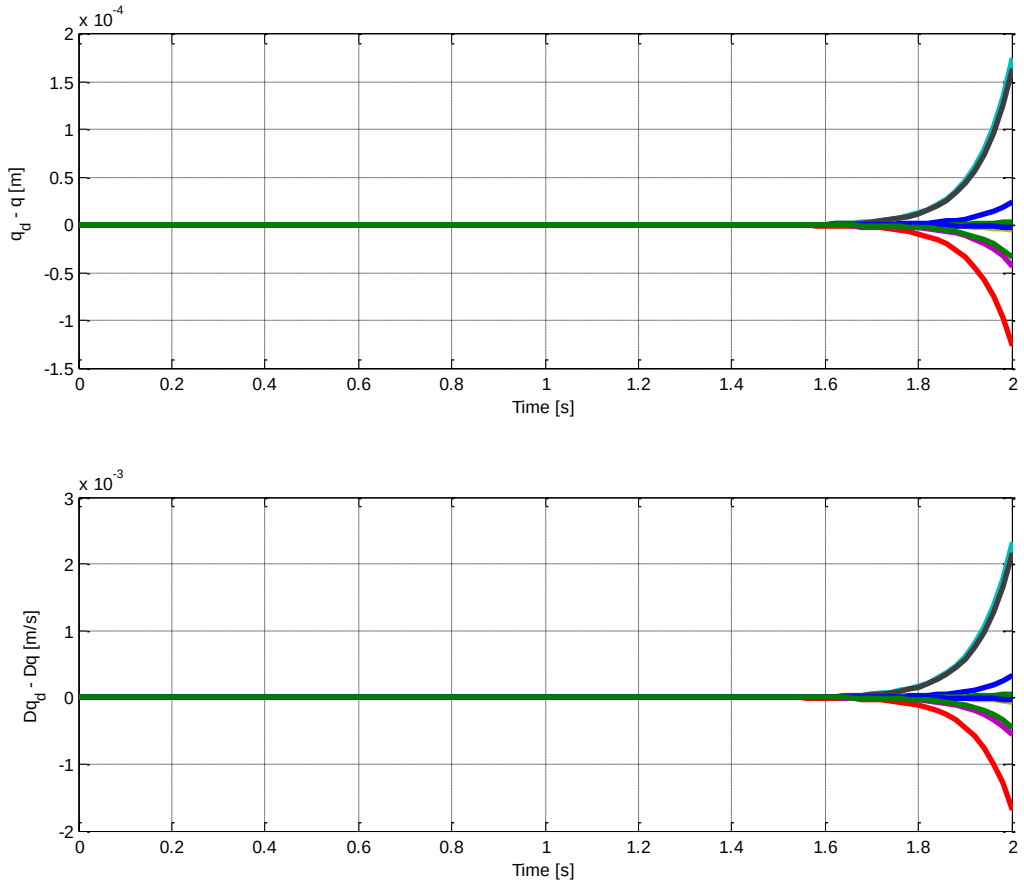

Figure 11: Error function of desired versus real position (top figure) and velocity (bottom figure)

For higher tolerances the error grows substantial closer to t_f; with this low tolerance the error at $t = t_f$ is of order 10^{-4} for position and 10^{-3} for the velocity. Figure 11 does not show a legend for coordinates, because it is not of importance in this figure. The goal of this figure is to show the substantially growing error due to the repeated numerical approach. Every time the approach produces a solution with a small error which is used in the next numerical approach. This way the small errors are accumulated and grow exponentially. But for this low tolerance the position and velocity of the computed evolution over two seconds match the desired trajectories (also the CoM matches). The fact the velocity and position match means also the acceleration and the ZMP should match. It can be concluded that with the actuator torques that the designed controller produces the desired trajectories are matched

4.6 Discussion

4.6.1 Difference with 3D

Apart from the fact that 3D is much more complicated than 2D and needs more computational time, there are more issues with 3D that have to be addresses to use the method explained in this study.

First of all, the matrices in the equation of motion grow a lot bigger, so it becomes more difficult to look at the separate expressions. The matrices can be linearly not independent. This can lead to problems in the decomposition as mentioned in Section 3.3.1 Maybe the decomposition can no longer be used or an extra constraint has to be added. This issue has to be investigated.

The parameter α which constraints the model and is calculated by the ZMP is not sufficient for 3D models, because the parameter only constrained the robot in the x-axis. For 3D, there is also the z-axis (if the y-axis is defined as vertical axis)that has to be constrained. Probably this can be done with an extra parameter β, which is calculated by the ZMP, which now also has a z-direction. So multiple artificial constraints should be used.

The inverse kinematics as calculated in Section 4.3 are quite simple. For 3D the two-link planar arm is not sufficient. The computations probably become so complicated that the inverse kinematics cannot be calculated algebraically, but a numerical algorithm has be used. Methods for this are already available in literature and can be implemented in this new method.

4.6.2 Feed-back

The feed-forward controller presented in this study works perfectly if the robot experiences no disturbances. The feed-forward controller would not supress any disturbance. In practice you will always encounter some disturbance, therefore a feed-back controller should be implemented together with a feed-forward controller. Possibly a simple PD-controller is already sufficient, but this has to be investigated. If the torques calculated as feed-forward are defined as τ_{ff}, the torque with the PD feed-back controller can be described by:

$$\tau = \tau_{ff} + K_d(\dot{q}_d - \dot{q}) + K_p(q_d - q) \tag{6.1}$$

This controller with feed-forward and feed-back calculates torques that smoothly changes contact forces from left to right foot and is quite robust against disturbances.

4.6.3 Trajectory

At the moment the hip is fixed in the vertical direction (y) and the movement was only in the x-direction. This is a very stiff walking motion, which is not very human-like. This walking motion can be improved by introducing also a motion in the y-direction. If an effective motion for y is designed with its velocity and acceleration, this could be implemented into the Matlab files quite simple presumably without any changes keeping in mind the ZMP should always be in between both feet. As mentioned in Section 2.2, the motion should be inside the null space of the constraints, (2.2) and (2.3) should hold.

6 Conclusions and recommendations

In this study methods to build a controller for the double support phase of a walking humanoid robot were investigated. After investigating an available method, a new method was designed based on decomposition of the general equation of motion with additional artificial constraints. Due to computational time and to avoid some 3-dimensional problems, a 2-dimensional example of a robot was used to evaluate this newly designed method. For this 2D-example, inverse kinematics and a third order trajectory were computed. Based on the Zero Moment Point (ZMP) a parameter α was introduced, which is used in the artificial constraints mentioned above. This parameter has a range between 0 and 1, which resembles the vertical contact forces that shift from the left foot to the right foot. With this method and a desired trajectory (position, velocity and acceleration) the contact forces at each point in the motion were calculated in Matlab and subsequently the torques in the joints were computed. With this set of torques and contact forces dynamic simulation was performed. With low simulation tolerance (10^{-12}) the real system evolution matched the desired trajectories (difference of order 10^{-4}).

This method successfully computed a feed-forward controller for double support that follows the desired trajectory and shifts the contact forces from the left leg to the right leg. In this study this method is only investigated on a 2D example. Some issues still have to be addressed for this method to be feasible in 3D, like the rank of the matrix J_c, the expansion of artificial constraints and the complexity of inverse kinematics in 3D. There are also some other topics that can improve the current controller, such as adding a feedback controller to reduce the effect of disturbances and producing a more human-like trajectory by not fixing the hip in vertical direction.

Appendix

A.1 Matlab-script of derivation of model parameters M, H, W and S

Made by Pieter van Zutven

```
% derive equations of motion of 7-link biped model with relative coordinates and virtual kinematic chain

clear all
close all
clc

%define variables and parameters, suitable for symbolic manipulation
syms m1 m2 m3 m4 m5 m6 m7
syms l1 l2 l3 l4 l5 l6 l7 lf lh
syms x y q1 q2 q3 q4 q5 q6 q7
syms Dx Dy Dq1 Dq2 Dq3 Dq4 Dq5 Dq6 Dq7
syms D2x D2y D2q1 D2q2 D2q3 D2q4 D2q5 D2q6 D2q7
syms g

m5 = m2;
m6 = m3;
m7 = m4;
l5 = l2;
l6 = l3;
l7 = l4;

c1 = l1/2;
c2 = l2/2;
c3 = l3/2;
c4 = l4/2;
c5 = l5/2;
c6 = l6/2;
c7 = l7/2;

%define vectors of independent states
qs   = [x;  y;  q1;  q2;  q3;  q4;  q5;  q6;  q7];
Dqs  = [Dx; Dy; Dq1; Dq2; Dq3; Dq4; Dq5; Dq6; Dq7];
D2qs = [D2x;D2y;D2q1;D2q2;D2q3;D2q4;D2q5;D2q6;D2q7];

%% Lagrange equations of motion for single support

% define the coordinates of all joints of the biped
%right hip
x1 = x;
y1 = y;

%right knee
x2 = x1+l2*sin(q1+q2);
y2 = y1+l2*cos(q1+q2);

%right ankle
x3 = x2+l3*sin(q1+q2+q3);
y3 = y2+l3*cos(q1+q2+q3);

%left hip
x4 = x1;
y4 = y1;

%left knee
x5 = x4+l5*sin(q1+q5);
y5 = y4+l5*cos(q1+q5);

%left ankle
x6 = x5+l6*sin(q1+q5+q6);
y6 = y5+l6*cos(q1+q5+q6);

% define the coordinates of all centers of mass of the biped
%torso
xs1 = x1+c1*sin(q1);
ys1 = y1+c1*cos(q1);
Dxs1 = jacobian(xs1,qs)*Dqs;
Dys1 = jacobian(ys1,qs)*Dqs;
```

```
%right upper leg
xs2 = x1+c2*sin(q1+q2);
ys2 = y1+c2*cos(q1+q2);
Dxs2 = jacobian(xs2,qs)*Dqs;
Dys2 = jacobian(ys2,qs)*Dqs;

%right lower leg
xs3 = x2+c3*sin(q1+q2+q3);
ys3 = y2+c3*cos(q1+q2+q3);
Dxs3 = jacobian(xs3,qs)*Dqs;
Dys3 = jacobian(ys3,qs)*Dqs;

%right foot
xs4 = x3+c4*sin(q1+q2+q3+q4);
ys4 = y3+c4*cos(q1+q2+q3+q4);
Dxs4 = jacobian(xs4,qs)*Dqs;
Dys4 = jacobian(ys4,qs)*Dqs;

%left upper leg
xs5 = x4+c5*sin(q1+q5);
ys5 = y4+c5*cos(q1+q5);
Dxs5 = jacobian(xs5,qs)*Dqs;
Dys5 = jacobian(ys5,qs)*Dqs;

%left lower leg
xs6 = x5+c6*sin(q1+q5+q6);
ys6 = y5+c6*cos(q1+q5+q6);
Dxs6 = jacobian(xs6,qs)*Dqs;
Dys6 = jacobian(ys6,qs)*Dqs;

%left foot
xs7 = x6+c7*sin(q1+q5+q6+q7);
ys7 = y6+c7*cos(q1+q5+q6+q7);
Dxs7 = jacobian(xs7,qs)*Dqs;
Dys7 = jacobian(ys7,qs)*Dqs;

kinetic energy
Ks = 1/2*m1*(Dxs1^2+Dys1^2) + 1/2*m2*(Dxs2^2+Dys2^2) + 1/2*m3*(Dxs3^2+Dys3^2) + 1/2*m4*(Dxs4^2+Dys4^2) + 1/2*m5*(Dxs5^2+Dys5^2) + 1/2*m6*(Dxs6^2+Dys6^2) + 1/2*m7*(Dxs7^2+Dys7^2)

%potential energy
Ps = m1*g*ys1 + m2*g*ys2 + m3*g*ys3 + m4*g*ys4 + m5*g*ys5 + m6*g*ys6 + m7*g*ys7

%Lagrange equation for single support
Ls = Ks-Ps;
taus = jacobian(jacobian(Ls,Dqs),[qs;Dqs])*[Dqs;D2qs]-jacobian(Ls,qs).'   %% jacobian(jacobian(Ls,Dqs),[qs;Dqs])*[Dqs;D2qs] == d/dt(T,qdot)

[Ds taus2] = matrize(taus,D2qs);
[Cs Gs] = matrize(taus2,Dqs);
check = simplify(taus-Ds*D2qs-Cs*Dqs-Gs)

CoM = 1/(m1+2*m2+2*m3+2*m4)*(m1*[xs1;ys1]+m2*[xs2;ys2]+m3*[xs3;ys3]+m4*[xs4;ys4]+m2*[xs5;ys5]+m3*[xs6;ys6]+m4*[xs7;ys7]);

matlabFunction(Ds,Cs,Gs,'file','continuousmodel')

%% Holonomic constraints

h = [x3+l4*sin(q1+q2+q3+q4);y3+l4*cos(q1+q2+q3+q4);y3-l4/2*cos(q1+q2+q3+q4);x6+l7*sin(q1+q5+q6+q7);y6+l7*cos(q1+q5+q6+q7);y6-l7/2*cos(q1+q5+q6+q7)];

W = simplify(jacobian(h,qs).');
w = simplify(jacobian(W.'*Dqs,qs)*Dqs);
matlabFunction(W,w,'file','doubleconstraint')
```

A.2 Matlab-script of derivation of velocity and acceleration

Made by Tom Dorussen

```
syms LB RB a1 a2 a3 a4 a5 l2 l3 t

x = a1+a2*t+a3*t^2+a4*t^3;
y = a5;
q1 = 0;

D1   = ((x-LB)^2+(y)^2-l2^2-l3^2)/(2*l2*l3);
sin1 = sqrt(1-D1^2);
cos1 = D1;
t1   = atan(y/(x-LB))-atan(l3*sin1/(l2+l3*cos1));
t2   = atan(sin1/cos1);
q2   = 1.5*pi-t1-t2;
q3   = t2;
q4   = 0.5*pi-q2-q3;

D2   = ((x-RB)^2+(y)^2-l2^2-l3^2)/(2*l2*l3);
sin2 = sqrt(1-D2^2);
cos2 = D2;
t3   = atan(y/(x-RB))-atan(l3*sin2/(l2+l3*cos2))+pi;
t4   = atan(sin2/cos2);
q5   = 0.5*pi-t3-t4;
q6   = t4;
q7   = 0.5*pi-q5-q6;

q = [x;y;q1;q2;q3;q4;q5;q6;q7];
Dx = jacobian(x,t);
Dy = jacobian(y,t);
Dq1 = jacobian(q1,t);
Dq2 = jacobian(q2,t);
Dq3 = jacobian(q3,t);
Dq4 = jacobian(q4,t);
Dq5 = jacobian(q5,t);
Dq6 = jacobian(q6,t);
Dq7 = jacobian(q7,t);

D2x = jacobian(Dx,t);
D2y = jacobian(Dy,t);
D2q1 = jacobian(Dq1,t);
D2q2 = jacobian(Dq2,t);
D2q3 = jacobian(Dq3,t);
D2q4 = jacobian(Dq4,t);
D2q5 = jacobian(Dq5,t);
D2q6 = jacobian(Dq6,t);
D2q7 = jacobian(Dq7,t);

q = [x;y;q1;q2;q3;q4;q5;q6;q7];
Dq = [Dx;Dy;Dq1;Dq2;Dq3;Dq4;Dq5;Dq6;Dq7];
D2q = [D2x;D2y;D2q1;D2q2;D2q3;D2q4;D2q5;D2q6;D2q7];

matlabFunction(Dq,D2q,'file','traj')
```

A.3 Matlab-script of calculation of controller

Made by Tom Dorussen

```
function Balanced_around_ZMP
clear all
clc
l1 = 0.5;
l2 = 0.5;
l3 = 0.5;
l4 = 0.1;
c1 = l1/2;
c2 = l2/2;
c3 = l3/2;
c4 = l4/2;
c5 = l2/2;
c6 = l3/2;
c7 = l4/2;
m1 = 5;
m2 = 2;
m3 = 1;
m4 = 0.5;
g = 9.81;

%% Create trajectory

% define the coordinates of all joints of the biped
%1 right hip
%2 right knee
%3 right ankle
%4 left hip
%5 left knee
%6 left ankle
tf = 2;
n = 100;
LB = -0.2;   % ZMP t0
RB = 0.2;    % ZMP tf
a = inv([1 0 0 0;0 1 0 0;1 tf tf^2 tf^3;0 1 2*tf 3*tf^2])*[LB-0.13;0;RB-0.08;0]; %Cubic polynomial trajectory
for i = 1:n+1
t(i)    = tf*(i-1)/n;
x(i)    = a(1)+a(2)*t(i)+a(3)*t(i)^2+a(4)*t(i)^3;   % 2nd order velocity profile
y(i)    = 0.7;                                      % Height Hip

a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);
a5 = y(i);

q1(i)   = 0;
D1(i)   = ((x(i)-LB)^2+(y(i))^2-l2^2-l3^2)/(2*l2*l3);
sin1(i) = sqrt(1-D1(i)^2);
cos1(i) = D1(i);
t1(i)   = atan2(y(i),x(i)-LB)-atan2(l3*sin1(i),l2+l3*cos1(i));
t2(i)   = atan2(sin1(i),cos1(i));
q2(i)   = 1.5*pi-t1(i)-t2(i);
q3(i)   = t2(i);
q4(i)   = 0.5*pi-q2(i)-q3(i);

D2(i)   = ((x(i)-RB)^2+(y(i))^2-l2^2-l3^2)/(2*l2*l3);
sin2(i) = sqrt(1-D2(i)^2);
cos2(i) = D2(i);
t3(i)   = atan2(y(i),x(i)-RB)-atan2(l3*sin2(i),l2+l3*cos2(i))+pi;
t4(i)   = atan2(sin2(i),cos2(i));
q5(i)   = 0.5*pi-t3(i)-t4(i);
q6(i)   = t4(i);
q7(i)   = 0.5*pi-q5(i)-q6(i);

q(:,i)  = [x(i);y(i);q1(i);q2(i);q3(i);q4(i);q5(i);q6(i);q7(i)];

[Dq(:,i),D2q(:,i)] = traj(LB,RB,a1,a2,a3,a4,a5,l2,l3,t(i));
% Dq(:,i) = zeros(9,1); D2q(:,i) = zeros(9,1);

Dx(i)  = Dq(1,i);
Dy(i)  = Dq(2,i);
```

```
Dq1(i) = Dq(3,i);
Dq2(i) = Dq(4,i);
Dq3(i) = Dq(5,i);
Dq4(i) = Dq(6,i);
Dq5(i) = Dq(7,i);
Dq6(i) = Dq(8,i);
Dq7(i) = Dq(9,i);

D2x(i)  = D2q(1,i);
D2y(i)  = D2q(2,i);
D2q1(i) = D2q(3,i);
D2q2(i) = D2q(4,i);
D2q3(i) = D2q(5,i);
D2q4(i) = D2q(6,i);
D2q5(i) = D2q(7,i);
D2q6(i) = D2q(8,i);
D2q7(i) = D2q(9,i);

x1(i) = x(i);
y1(i) = y(i);
x2(i) = x(i)+l2*sin(q1(i)+q2(i));
y2(i) = y(i)+l2*cos(q1(i)+q2(i));
x3(i) = x(i)+l2*sin(q1(i)+q2(i))+l3*sin(q1(i)+q2(i)+q3(i));
y3(i) = y(i)+l2*cos(q1(i)+q2(i))+l3*cos(q1(i)+q2(i)+q3(i));
x4(i) = x(i);
y4(i) = y(i);
x5(i) = x(i)+l2*sin(q1(i)+q5(i));
y5(i) = y(i)+l2*cos(q1(i)+q5(i));
x6(i) = x(i)+l2*sin(q1(i)+q5(i))+l3*sin(q1(i)+q5(i)+q6(i));
y6(i) = y(i)+l2*cos(q1(i)+q5(i))+l3*cos(q1(i)+q5(i)+q6(i));

[M(:,:,i),C,G]                                                                                                                                  =
continuousmodel(Dq1(i),Dq2(i),Dq3(i),Dq4(i),Dq5(i),Dq6(i),Dq7(i),g,l1,l2,l3,l4,m1,m2,m3,m4,q1(i),q2(i),q3(i),q4(i),q5(i),q6(i),q7(i)
);
[W,w] = doubleconstraint(Dq1(i),Dq2(i),Dq3(i),Dq4(i),Dq5(i),Dq6(i),Dq7(i),l2,l3,l4,q1(i),q2(i),q3(i),q4(i),q5(i),q6(i),q7(i));

Jc(:,:,i) = W';
S(:,:,i) = [zeros(3,6);eye(6)];
H(:,i) = C*Dq(:,i)+G;
%% Decompositie
S1      = [eye(3);zeros(6,3)];
S2      = S(:,:,i);
M1      = S1'*M(:,:,i);
H1      = S1'*H(:,i);
Jc1     = Jc(:,:,i)*S1;
M2      = S2'*M(:,:,i);
H2      = S2'*H(:,i);
Jc2     = Jc(:,:,i)*S2;

%% Standard function
% lambda  = pinv(Jc1')*(M1*D2q+H1);
% tau     = M2*D2q+H2-Jc2'*lambda;
% check   = M*D2q+H-S*tau-Jc'*lambda;

%% ZMP and alfa
CoMx(i) = (m3*(x(i) + l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2) + m3*(x(i) + l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i)
+ q6(i)))/2) + m2*(x(i) + (l2*sin(q1(i) + q2(i)))/2) + m2*(x(i) + (l2*sin(q1(i) + q5(i)))/2) + m1*(x(i) + (l1*sin(q1(i)))/2) + m4*(x(i) +
(l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*sin(q1(i) + q2(i)) + l3*sin(q1(i) + q2(i) + q3(i))) + m4*(x(i) + (l4*sin(q1(i) + q5(i) + q6(i) +
q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i))))/(m1 + 2*m2 + 2*m3 + 2*m4);
ZMPx(i) = (m4*((x(i) + (l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*sin(q1(i) + q2(i)) + l3*sin(q1(i) + q2(i) +
q3(i)))*(D2q3(i)*((l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l3*sin(q1(i) + q2(i) + q3(i))) - g - D2y(i) + D2q1(i)*((l4*sin(q1(i) + q2(i) +
q3(i) + q4(i)))/2 + l2*sin(q1(i) + q2(i)) + l3*sin(q1(i) + q2(i) + q3(i))) + D2q2(i)*((l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*sin(q1(i)
+ q2(i)) + l3*sin(q1(i) + q2(i) + q3(i))) + (D2q4(i)*l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2) + (y(i) + (l4*cos(q1(i) + q2(i) + q3(i) +
q4(i)))/2 + l2*cos(q1(i) + q2(i)) + l3*cos(q1(i) + q2(i) + q3(i)))*(D2x(i) + D2q3(i)*((l4*cos(q1(i) + q2(i) + q3(i) + q4(i)))/2 +
l3*cos(q1(i) + q2(i) + q3(i))) + D2q1(i)*((l4*cos(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*cos(q1(i) + q2(i)) + l3*cos(q1(i) + q2(i) + q3(i)))
+ D2q2(i)*((l4*cos(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*cos(q1(i) + q2(i)) + l3*cos(q1(i) + q2(i) + q3(i))) + (D2q4(i)*l4*cos(q1(i) +
q2(i) + q3(i) + q4(i)))/2)) - m2*((x(i) + (l2*sin(q1(i) + q5(i)))/2)*(D2y(i) + g - (D2q1(i)*l2*sin(q1(i) + q5(i)))/2 - (D2q5(i)*l2*sin(q1(i) +
q5(i)))/2) - (y(i) + (l2*cos(q1(i) + q5(i)))/2)*(D2x(i) + (D2q1(i)*l2*cos(q1(i) + q5(i)))/2 + (D2q5(i)*l2*cos(q1(i) + q5(i)))/2)) - m2*((x(i)
+ (l2*sin(q1(i) + q2(i)))/2)*(D2y(i) + g - (D2q1(i)*l2*sin(q1(i) + q2(i)))/2 - (D2q2(i)*l2*sin(q1(i) + q2(i)))/2) - (y(i) + (l2*cos(q1(i) +
q2(i)))/2)*(D2x(i) + (D2q1(i)*l2*cos(q1(i) + q2(i)))/2 + (D2q2(i)*l2*cos(q1(i) + q2(i)))/2)) + m4*((x(i) + (l4*sin(q1(i) + q5(i) + q6(i) +
q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i)))*(D2q6(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l3*sin(q1(i) + q5(i)
+ q6(i))) - g - D2y(i) + D2q1(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i))) +
D2q5(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i))) + (D2q7(i)*l4*sin(q1(i) + q5(i) +
q6(i) + q7(i)))/2) + (y(i) + (l4*cos(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*cos(q1(i) + q5(i)) + l3*cos(q1(i) + q5(i) + q6(i)))*(D2x(i) +
D2q6(i)*((l4*cos(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l3*cos(q1(i) + q5(i) + q6(i))) + D2q1(i)*((l4*cos(q1(i) + q5(i) + q6(i) + q7(i)))/2 +
```

```
l2*cos(q1(i) + q5(i)) + l3*cos(q1(i) + q5(i) + q6(i))) + D2q5(i)*((l4*cos(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*cos(q1(i) + q5(i)) + l3*cos(q1(i) + q5(i) + q6(i))) + (D2q7(i)*l4*cos(q1(i) + q5(i) + q6(i) + q7(i)))/2)) + m1*((y(i) + (l1*cos(q1(i)))/2)*(D2x(i) + (D2q1(i)*l1*cos(q1(i)))/2) - (x(i) + (l1*sin(q1(i)))/2)*(D2y(i) + g - (D2q1(i)*l1*sin(q1(i)))/2)) + m3*((x(i) + l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2)*(D2q1(i)*(l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2) - g - D2y(i) + D2q2(i)*(l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2) + (D2q3(i)*l3*sin(q1(i) + q2(i) + q3(i)))/2) + (y(i) + l2*cos(q1(i) + q2(i)) + (l3*cos(q1(i) + q2(i) + q3(i)))/2)*(D2x(i) + D2q1(i)*(l2*cos(q1(i) + q2(i)) + (l3*cos(q1(i) + q2(i) + q3(i)))/2) + D2q2(i)*(l2*cos(q1(i) + q2(i)) + (l3*cos(q1(i) + q2(i) + q3(i)))/2) + (D2q3(i)*l3*cos(q1(i) + q2(i) + q3(i)))/2)) + m3*((x(i) + l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i) + q6(i)))/2)*(D2q1(i)*(l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i) + q6(i)))/2) - g - D2y(i) + D2q5(i)*(l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i) + q6(i)))/2) + (D2q6(i)*l3*sin(q1(i) + q5(i) + q6(i)))/2) + (y(i) + l2*cos(q1(i) + q5(i)) + (l3*cos(q1(i) + q5(i) + q6(i)))/2)*(D2x(i) + D2q1(i)*(l2*cos(q1(i) + q5(i)) + (l3*cos(q1(i) + q5(i) + q6(i)))/2) + D2q5(i)*(l2*cos(q1(i) + q5(i)) + (l3*cos(q1(i) + q5(i) + q6(i)))/2) + (D2q6(i)*l3*cos(q1(i) + q5(i) + q6(i)))/2)))/(m4*(D2q3(i)*((l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l3*sin(q1(i) + q2(i) + q3(i))) - g - D2y(i) + D2q1(i)*((l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*sin(q1(i) + q2(i)) + l3*sin(q1(i) + q2(i) + q3(i))) + D2q2(i)*((l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2 + l2*sin(q1(i) + q2(i)) + l3*sin(q1(i) + q2(i) + q3(i))) + (D2q4(i)*l4*sin(q1(i) + q2(i) + q3(i) + q4(i)))/2) + m4*(D2q6(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l3*sin(q1(i) + q5(i) + q6(i))) - g - D2y(i) + D2q1(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i))) + D2q5(i)*((l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2 + l2*sin(q1(i) + q5(i)) + l3*sin(q1(i) + q5(i) + q6(i))) + (D2q7(i)*l4*sin(q1(i) + q5(i) + q6(i) + q7(i)))/2) - m2*(D2y(i) + g - (D2q1(i)*l2*sin(q1(i) + q2(i)))/2 - (D2q2(i)*l2*sin(q1(i) + q2(i)))/2) - m2*(D2y(i) + g - (D2q1(i)*l2*sin(q1(i) + q5(i)))/2 - (D2q5(i)*l2*sin(q1(i) + q5(i)))/2) + m3*(D2q1(i)*(l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2) - g - D2y(i) + D2q2(i)*(l2*sin(q1(i) + q2(i)) + (l3*sin(q1(i) + q2(i) + q3(i)))/2) + (D2q3(i)*l3*sin(q1(i) + q2(i) + q3(i)))/2) + m3*(D2q1(i)*(l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i) + q6(i)))/2) - g - D2y(i) + D2q5(i)*(l2*sin(q1(i) + q5(i)) + (l3*sin(q1(i) + q5(i) + q6(i)))/2) + (D2q6(i)*l3*sin(q1(i) + q5(i) + q6(i)))/2) - m1*(D2y(i) + g - (D2q1(i)*l1*sin(q1(i)))/2));

alfa(i) = (ZMPx(i)-LB)/(RB-LB);
if alfa(i) < 0
    alfa(i) = 0;   % Total weight on 1st leg
elseif alfa(i) > 1
    alfa(i) = 1;   % Total weight on 2nd leg
end
%% Adding constraints
C1 = [0 -alfa(i) 0 0 1-alfa(i) 0];
C2 = [0 0 -alfa(i) 0 0 1-alfa(i)];
C3 = [0 0 0 0 0 0];
A  = [Jc1';C1;C2;C3];
b  = [M1*D2q(:,i)+H1;0;0;0];
lambda(:,i) = pinv(A)*b;
tau(:,i)    = M2*D2q(:,i)+H2-Jc2'*lambda(:,i);
check(:,i)  = M(:,:,i)*D2q(:,i)+H(:,i)-S(:,:,i)*tau(:,i)-Jc(:,:,i)'*lambda(:,i);
end
%% Calcalating trajectory from the torques
Q0      = [q(:,1);Dq(:,1)];
TSPAN = t;
options = odeset('RelTol',1e-12,'AbsTol',1e-12*ones(18,1));
[T,Q]   = ode45('FUN2',TSPAN,Q0,options);
figure;
subplot(2,2,1); plot(T,Q(:,1:9)); grid on
subplot(2,2,2); plot(T,q); grid on
subplot(2,2,3); plot(t,Q(:,10:18)); grid on
subplot(2,2,4); plot(t,Dq); grid on


%% Plot figures
figure;
subplot(3,1,1); plot(t,q); grid on
subplot(3,1,2); plot(t,Dq); grid on
subplot(3,1,3); plot(t,D2q); grid on
figure; plot(t,tau); grid on
figure; plot(t,lambda); grid on
%% Walking
% visualize(t,Q(:,1:9),'play',@drawrobot)
% visualize(t,q','play',@drawrobot)
% %assign all variables in base workspace
% cellfun(@(vars) assignin('base',vars,eval(vars)),who,'ErrorHandler',@(~,~) disp(''));
%
% % draw robot for animation function
%     function hndls = drawrobot(~,xviz,hndls)
%
%         if isempty(hndls)
%             %initialization
%             hndls(1) = figure('Color',[1 1 1]);
%
%             axis([-1 1 -0.5 1.6])
%             axis equal
%             axis manual
%
%             hndls(2) = line('Color',[0 0 0],'LineWidth',2,'Marker','o','MarkerSize',2);
```

```
%           hndls(3) = line('Color',[1 0 0],'LineWidth',2,'Marker','o','MarkerSize',2);
%           hndls(4) = line('Color',[1 0 0],'LineWidth',2,'Marker','o','MarkerSize',2);
%           hndls(5) = line('Color',[0 0 1],'LineWidth',2,'Marker','o','MarkerSize',2);
%           hndls(6) = line('Color',[0 0 1],'LineWidth',2,'Marker','o','MarkerSize',2);
%           hndls(7) = line('Color',[0 1 0],'LineStyle','none','Marker','*','MarkerSize',10);
%           hndls(8) = line('Color',[0 0 0],'LineStyle',':','Marker','.','MarkerSize',10);
%       end
%
%       qviz = xviz(1:9);
%
%       xviz = qviz(1);
%       yviz = qviz(2);
%       qviz1 = qviz(3);
%       qviz2 = qviz(4);
%       qviz3 = qviz(5);
%       qviz4 = qviz(6);
%       qviz5 = qviz(7);
%       qviz6 = qviz(8);
%       qviz7 = qviz(9);
%
%       %torso
%       x0 = xviz+l1*sin(qviz1);
%       y0 = yviz+l1*cos(qviz1);
%
%       %right hip
%       x1 = xviz;
%       y1 = yviz;
%
%       %right knee
%       x2 = x1+l2*sin(qviz1+qviz2);
%       y2 = y1+l2*cos(qviz1+qviz2);
%
%       %right ankle
%       x3 = x2+l3*sin(qviz1+qviz2+qviz3);
%       y3 = y2+l3*cos(qviz1+qviz2+qviz3);
%
%       %left hip
%       x4 = x1;
%       y4 = y1;
%
%       %left knee
%       x5 = x4+l2*sin(qviz1+qviz5);
%       y5 = y4+l2*cos(qviz1+qviz5);
%
%       %left ankle
%       x6 = x5+l3*sin(qviz1+qviz5+qviz6);
%       y6 = y5+l3*cos(qviz1+qviz5+qviz6);
%
%       %right toe
%       x3t = x3+l4*sin(qviz1+qviz2+qviz3+qviz4);
%       y3t = y3+l4*cos(qviz1+qviz2+qviz3+qviz4);
%
%       %right heel
%       x3h = x3-l4/2*sin(qviz1+qviz2+qviz3+qviz4);
%       y3h = y3-l4/2*cos(qviz1+qviz2+qviz3+qviz4);
%
%       %left toe
%       x6t = x6+l4*sin(qviz1+qviz5+qviz6+qviz7);
%       y6t = y6+l4*cos(qviz1+qviz5+qviz6+qviz7);
%
%       %left heel
%       x6h = x6-l4/2*sin(qviz1+qviz5+qviz6+qviz7);
%       y6h = y6-l4/2*cos(qviz1+qviz5+qviz6+qviz7);
%
%       pos = [x0 x1 x2 x3 x3t x3h x4 x5 x6 x6t x6h;y0 y1 y2 y3 y3t y3h y4 y5 y6 y6t y6h];
%
%       %torso
%       xs1 = x1+c1*sin(qviz1);
%       ys1 = y1+c1*cos(qviz1);
%
%       %right upper leg
%       xs2 = x1+c2*sin(qviz1+qviz2);
%       ys2 = y1+c2*cos(qviz1+qviz2);
%
%       %right lower leg
```

```
%         xs3 = x2+c3*sin(qviz1+qviz2+qviz3);
%         ys3 = y2+c3*cos(qviz1+qviz2+qviz3);
%
%         %right foot
%         xs4 = x3+c4*sin(qviz1+qviz2+qviz3+qviz4);
%         ys4 = y3+c4*cos(qviz1+qviz2+qviz3+qviz4);
%
%         %left upper leg
%         xs5 = x4+c5*sin(qviz1+qviz5);
%         ys5 = y4+c5*cos(qviz1+qviz5);
%
%         %left lower leg
%         xs6 = x5+c6*sin(qviz1+qviz5+qviz6);
%         ys6 = y5+c6*cos(qviz1+qviz5+qviz6);
%
%         %left foot
%         xs7 = x6+c7*sin(qviz1+qviz5+qviz6+qviz7);
%         ys7 = y6+c7*cos(qviz1+qviz5+qviz6+qviz7);
%
%         poscom = [xs1 xs2 xs3 xs4 xs5 xs6 xs7;ys1 ys2 ys3 ys4 ys5 ys6 ys7];
%
%                                                                                                                 posCOM                  =
1/(m1+2*m2+2*m3+2*m4)*(m1*[xs1;ys1]+m2*[xs2;ys2]+m3*[xs3;ys3]+m4*[xs4;ys4]+m2*[xs5;ys5]+m3*[xs6;ys6]+m4*[xs7;ys7])
;
%
%         %draw all lines
%         set(hndls(2),'Xdata',pos(1,1:2),'Ydata',pos(2,1:2))
%         set(hndls(3),'Xdata',pos(1,2:4),'Ydata',pos(2,2:4))
%         set(hndls(4),'Xdata',pos(1,5:6),'Ydata',pos(2,5:6))
%         set(hndls(5),'Xdata',pos(1,7:9),'Ydata',pos(2,7:9))
%         set(hndls(6),'Xdata',pos(1,10:11),'Ydata',pos(2,10:11))
%         set(hndls(7),'Xdata',poscom(1,:),'Ydata',poscom(2,:))
%         set(hndls(8),'Xdata',[posCOM(1) posCOM(1)],'Ydata',[posCOM(2) 0])
%     end

end
```

A.4 Matlab-scipt of ode45 function

Made by Tom Dorussen

```
function Xdot = FUN2(T,X)
Q = X(1:9);
DQ = X(10:18);
%% Calculate tau
l1 = 0.5;
l2 = 0.5;
l3 = 0.5;
l4 = 0.1;
c1 = l1/2;
c2 = l2/2;
c3 = l3/2;
c4 = l4/2;
c5 = l2/2;
c6 = l3/2;
c7 = l4/2;
m1 = 5;
m2 = 2;
m3 = 1;
m4 = 0.5;
g = 9.81;
% Create trajectory
tf = 2;
LB = -0.2;   % ZMP t0
RB = 0.2;    % ZMP tf
a = inv([1 0 0 0;0 1 0 0;1 tf tf^2 tf^3;0 1 2*tf 3*tf^2])*[LB-0.13;0;RB-0.08;0]; %Cubic polynomial trajectory
x    = a(1)+a(2)*T+a(3)*T^2+a(4)*T^3;   % 2nd order velocity profile
y    = 0.7;                             % Height Hip
a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);
a5 = y;
q1   = 0;
D1   = ((x-LB)^2+(y)^2-l2^2-l3^2)/(2*l2*l3);
sin1 = sqrt(1-D1^2);
cos1 = D1;
t1   = atan2(y,x-LB)-atan2(l3*sin1,l2+l3*cos1);
t2   = atan2(sin1,cos1);
q2   = 1.5*pi-t1-t2;
q3   = t2;
q4   = 0.5*pi-q2-q3;
D2   = ((x-RB)^2+(y)^2-l2^2-l3^2)/(2*l2*l3);
sin2 = sqrt(1-D2^2);
cos2 = D2;
t3   = atan2(y,x-RB)-atan2(l3*sin2,l2+l3*cos2)+pi;
t4   = atan2(sin2,cos2);
q5   = 0.5*pi-t3-t4;
q6   = t4;
q7   = 0.5*pi-q5-q6;
q  = [x;y;q1;q2;q3;q4;q5;q6;q7];
[Dq,D2q] = traj(LB,RB,a1,a2,a3,a4,a5,l2,l3,T);
Dx  = Dq(1);
Dy  = Dq(2);
Dq1 = Dq(3);
Dq2 = Dq(4);
Dq3 = Dq(5);
Dq4 = Dq(6);
Dq5 = Dq(7);
Dq6 = Dq(8);
Dq7 = Dq(9);
D2x  = D2q(1);
D2y  = D2q(2);
D2q1 = D2q(3);
D2q2 = D2q(4);
D2q3 = D2q(5);
D2q4 = D2q(6);
D2q5 = D2q(7);
D2q6 = D2q(8);
D2q7 = D2q(9);
[M,C,G] = continuousmodel(Dq1,Dq2,Dq3,Dq4,Dq5,Dq6,Dq7,g,l1,l2,l3,l4,m1,m2,m3,m4,q1,q2,q3,q4,q5,q6,q7);
[W,w] = doubleconstraint(Dq1,Dq2,Dq3,Dq4,Dq5,Dq6,Dq7,l2,l3,l4,q1,q2,q3,q4,q5,q6,q7);
```

```
Jc = W';
S = [zeros(3,6);eye(6)];
H = C*Dq+G;
% Decompositie
S1      = [eye(3);zeros(6,3)];
S2      = S;
M1      = S1'*M;
H1      = S1'*H;
Jc1     = Jc*S1;
M2      = S2'*M;
H2      = S2'*H;
Jc2     = Jc*S2;
% ZMP
ZMPx = (m4*((x + (l4*sin(q1 + q2 + q3 + q4))/2 + l2*sin(q1 + q2) + l3*sin(q1 + q2 + q3))*(D2q3*((l4*sin(q1 + q2 + q3 + q4))/2 +
l3*sin(q1 + q2 + q3)) - g - D2y + D2q1*((l4*sin(q1 + q2 + q3 + q4))/2 + l2*sin(q1 + q2) + l3*sin(q1 + q2 + q3)) + D2q2*((l4*sin(q1
+ q2 + q3 + q4))/2 + l2*sin(q1 + q2) + l3*sin(q1 + q2 + q3)) + (D2q4*l4*sin(q1 + q2 + q3 + q4))/2) + (y + (l4*cos(q1 + q2 + q3 +
q4))/2 + l2*cos(q1 + q2) + l3*cos(q1 + q2 + q3))*(D2x + D2q3*((l4*cos(q1 + q2 + q3 + q4))/2 + l3*cos(q1 + q2 + q3)) +
D2q1*((l4*cos(q1 + q2 + q3 + q4))/2 + l2*cos(q1 + q2) + l3*cos(q1 + q2 + q3)) + D2q2*((l4*cos(q1 + q2 + q3 + q4))/2 +
l2*cos(q1 + q2) + l3*cos(q1 + q2 + q3)) + (D2q4*l4*cos(q1 + q2 + q3 + q4))/2)) - m2*((x + (l2*sin(q1 + q5))/2)*(D2y + g -
(D2q1*l2*sin(q1 + q5))/2 - (D2q5*l2*sin(q1 + q5))/2) - (y + (l2*cos(q1 + q5))/2)*(D2x + (D2q1*l2*cos(q1 + q5))/2 +
(D2q5*l2*cos(q1 + q5))/2)) - m2*((x + (l2*sin(q1 + q2))/2)*(D2y + g - (D2q1*l2*sin(q1 + q2))/2 - (D2q2*l2*sin(q1 + q2))/2) - (y +
(l2*cos(q1 + q2))/2)*(D2x + (D2q1*l2*cos(q1 + q2))/2 + (D2q2*l2*cos(q1 + q2))/2)) + m4*((x + (l4*sin(q1 + q5 + q6 + q7))/2 +
l2*sin(q1 + q5) + l3*sin(q1 + q5 + q6))*(D2q6*((l4*sin(q1 + q5 + q6 + q7))/2 + l3*sin(q1 + q5 + q6)) - g - D2y + D2q1*((l4*sin(q1
+ q5 + q6 + q7))/2 + l2*sin(q1 + q5) + l3*sin(q1 + q5 + q6)) + D2q5*((l4*sin(q1 + q5 + q6 + q7))/2 + l2*sin(q1 + q5) + l3*sin(q1 +
q5 + q6)) + (D2q7*l4*sin(q1 + q5 + q6 + q7))/2) + (y + (l4*cos(q1 + q5 + q6 + q7))/2 + l2*cos(q1 + q5) + l3*cos(q1 + q5 +
q6))*(D2x + D2q6*((l4*cos(q1 + q5 + q6 + q7))/2 + l3*cos(q1 + q5 + q6)) + D2q1*((l4*cos(q1 + q5 + q6 + q7))/2 + l2*cos(q1 +
q5) + l3*cos(q1 + q5 + q6)) + D2q5*((l4*cos(q1 + q5 + q6 + q7))/2 + l2*cos(q1 + q5) + l3*cos(q1 + q5 + q6)) + (D2q7*l4*cos(q1
+ q5 + q6 + q7))/2)) + m1*((y + (l1*cos(q1))/2)*(D2x + (D2q1*l1*cos(q1))/2) - (x + (l1*sin(q1))/2)*(D2y + g - (D2q1*l1*sin(q1))/2))
+ m3*((x + l2*sin(q1 + q2) + (l3*sin(q1 + q2 + q3))/2)*(D2q1*(l2*sin(q1 + q2) + (l3*sin(q1 + q2 + q3))/2) - g - D2y +
D2q2*(l2*sin(q1 + q2) + (l3*sin(q1 + q2 + q3))/2) + (D2q3*l3*sin(q1 + q2 + q3))/2) + (y + l2*cos(q1 + q2) + (l3*cos(q1 + q2 +
q3))/2)*(D2x + D2q1*(l2*cos(q1 + q2) + (l3*cos(q1 + q2 + q3))/2) + D2q2*(l2*cos(q1 + q2) + (l3*cos(q1 + q2 + q3))/2) +
(D2q3*l3*cos(q1 + q2 + q3))/2)) + m3*((x + l2*sin(q1 + q5) + (l3*sin(q1 + q5 + q6))/2)*(D2q1*(l2*sin(q1 + q5) + (l3*sin(q1 + q5 +
q6))/2) - g - D2y + D2q5*(l2*sin(q1 + q5) + (l3*sin(q1 + q5 + q6))/2) + (D2q6*l3*sin(q1 + q5 + q6))/2) + (y + l2*cos(q1 + q5) +
(l3*cos(q1 + q5 + q6))/2)*(D2x + D2q1*(l2*cos(q1 + q5) + (l3*cos(q1 + q5 + q6))/2) + D2q5*(l2*cos(q1 + q5) + (l3*cos(q1 + q5 +
q6))/2) + (D2q6*l3*cos(q1 + q5 + q6))/2)))/(m4*(D2q3*((l4*sin(q1 + q2 + q3 + q4))/2 + l3*sin(q1 + q2 + q3)) - g - D2y +
D2q1*((l4*sin(q1 + q2 + q3 + q4))/2 + l2*sin(q1 + q2) + l3*sin(q1 + q2 + q3)) + D2q2*((l4*sin(q1 + q2 + q3 + q4))/2 + l2*sin(q1 +
q2) + l3*sin(q1 + q2 + q3)) + (D2q4*l4*sin(q1 + q2 + q3 + q4))/2) + m4*(D2q6*((l4*sin(q1 + q5 + q6 + q7))/2 + l3*sin(q1 + q5 +
q6)) - g - D2y + D2q1*((l4*sin(q1 + q5 + q6 + q7))/2 + l2*sin(q1 + q5) + l3*sin(q1 + q5 + q6)) + D2q5*((l4*sin(q1 + q5 + q6 +
q7))/2 + l2*sin(q1 + q5) + l3*sin(q1 + q5 + q6)) + (D2q7*l4*sin(q1 + q5 + q6 + q7))/2) - m2*(D2y + g - (D2q1*l2*sin(q1 + q2))/2 -
(D2q2*l2*sin(q1 + q2))/2) - m2*(D2y + g - (D2q1*l2*sin(q1 + q5))/2 - (D2q5*l2*sin(q1 + q5))/2) + m3*(D2q1*(l2*sin(q1 + q2) +
(l3*sin(q1 + q2 + q3))/2) - g - D2y + D2q2*(l2*sin(q1 + q2) + (l3*sin(q1 + q2 + q3))/2) + (D2q3*l3*sin(q1 + q2 + q3))/2) +
m3*(D2q1*(l2*sin(q1 + q5) + (l3*sin(q1 + q5 + q6))/2) - g - D2y + D2q5*(l2*sin(q1 + q5) + (l3*sin(q1 + q5 + q6))/2) +
(D2q6*l3*sin(q1 + q5 + q6))/2) - m1*(D2y + g - (D2q1*l1*sin(q1))/2));
alfa = (ZMPx-LB)/(RB-LB);
if alfa < 0
    alfa = 0;   % Total weight on 1st leg
elseif alfa > 1
    alfa = 1;   % Total weight on 2nd leg
end
% Adding constraints
C1 = [0 -alfa 0 0 1-alfa 0];
C2 = [0 0 -alfa 0 0 1-alfa];
C3 = [0 0 0 0 0 0];
A  = [Jc1';C1;C2;C3];
b  = [M1*D2q+H1;0;0;0];
lambda = pinv(A)*b;
tau    = M2*D2q+H2-Jc2'*lambda;
%% Forward-dynamics
% M,H,S,Jc
[Mc,Cc,Gc]                                                                                                    =
continuousmodel(DQ(3),DQ(4),DQ(5),DQ(6),DQ(7),DQ(8),DQ(9),g,l1,l2,l3,l4,m1,m2,m3,m4,Q(3),Q(4),Q(5),Q(6),Q(7),Q(8),Q(9)
);
[Wc,wc]    = doubleconstraint(DQ(3),DQ(4),DQ(5),DQ(6),DQ(7),DQ(8),DQ(9),l2,l3,l4,Q(3),Q(4),Q(5),Q(6),Q(7),Q(8),Q(9));
Sc         = [zeros(3,6);eye(6)];
Hc         = Cc*DQ+Gc;
% lambda en D2Q
lambdac   = inv(Wc'*inv(Mc)*Wc)*(Wc'*inv(Mc)*(Hc-Sc*tau)-wc);
D2Q       = inv(Mc)*(Sc*tau+Wc*lambdac-Hc);
Xdot      = [DQ;D2Q];
end
```

References

[1] *TechUnited Eindhoven* (no data), Regularly updated. [cited 1 Aug. 2013]. Available from World Wide Web: <http://www.techunited.nl/nl/tulip/>

[2] Van Zutven, P. (2013), , *Master open space assignment: Model based dynamics compensation for humanoid robots*, Eindhoven, University of Technology, Department of Mechanical Engineering

[3] Van der Wouw, N. (2012), Dynamics of Multibody Systems, *Multibody Dynamics*, University Press Facilities, Eindhoven, p 99-127

[4] Van Zutven P. (2009), *Modeling, Identification and Stability of Humanoid Robots*, Master Thesis, Eindhoven, University of Technology, Department of Mechanical Engineering ,p. 1

[5] Baelemans, J.A.J. (2013) , *Parameter Estimation of humanoid robots using the center of pressure*, Eindhoven, University of Technology, Department of Mechanical Engineering, Available from World Wide Web:<http://www.techunited.nl/media/files/humanoid/JosBaelemans_ GRAD2013_Parameter _estimation_of_humanoid_robots_using_the_center_of_pressure.pdf?

[6] *Robocup Eindhoven 2013* (2013), Available from World Wide Web:
< http://www.robocup2013.org/humanoid-adult-size-schedule/?lang=nl>

[7] Spong, M.W.; Hutchinson, S.; Vidyasagar, M., (2006), *The Denavit-Hartenberg Convention*, *Robot Modeling and Control*, Wiley, New York, ISBN 978-0-471-64990-8, p. 76-92

[8] Mistry, M.; Buchli, J.; Schaal, S. (2010), *Inverse Dynamics Control of Floating Base Systems using Orthogonal Decomposition*, Anchorage, Alaska

[9] Dekker M.H.P (2009), *Zero-Moment Point Method for Stable Biped Walking*, Eindhoven, University of Technology, Department of Mechanical Engineering

[10] Spong, M.W.; Hutchinson, S.; Vidyasagar, M., (2006), *Trajectory planning*, *Robot Modeling and Control*, Wiley, New York, ISBN 978-0-471-64990-8, p. 186-198

[11] Spong, M.W.; Hutchinson, S.; Vidyasagar, M., (2006), *Inverse Kinematics*, *Robot Modeling and Control*, Wiley, New York, ISBN 978-0-471-64990-8, p. 93-109